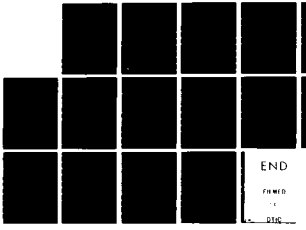
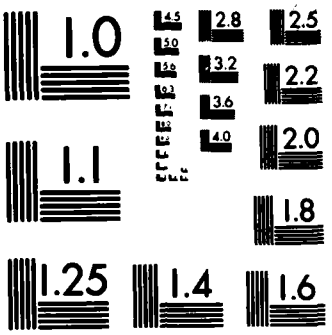
AD-A131 306 RESEARCH ON INTERACTIVE ACQUISITION AND USE OF KNOWLEDGE(U) SRI INTERNATIONAL MENLO PARK CA M E STICKEL MAY 83 N00039-80-C-0575 1/1

UNCLASSIFIED F/G 9/2 NL





SRI International

RESEARCH ON INTERACTIVE ACQUISITION AND USE OF KNOWLEDGE

Interim Report for the Period July 1982–January 1983

SRI Project 1894

May 1983

Edited by: Mark E. Stickel, Senior Computer Scientist
Artificial Intelligence Center
Computer Science and Technology Division

Prepared for:

Defense Advanced Research Projects Agency
Information Processing Techniques Office
1400 Wilson Boulevard
Arlington, Virginia 22209

Attention: Cmdr. Ronald B. Ohlander

Preparation of this paper was supported by the Defense Advanced Research Projects Agency under Contract N00039-80-C-0575 with the Naval Electronic Systems Command.

SRI International
333 Ravenswood Ave. • Menlo Park, CA 94025
(415) 859-6200 • TWX: 910-373-2046 • Telex: 334 486

Contents

1. Introduction

SRI International is engaged in a long-term effort under DARPA sponsorship to conduct basic research on artificial intelligence problems central to the construction of computer systems that can participate in extended dialogues in natural language with their users. Such dialogue capabilities are crucial to providing systems that can interact with users, providing information about the system's capabilities and knowledge and aiding the user in solving problems requiring information held by the system. A central thrust to our research is that the system be able to acquire new concepts, facts, and vocabulary through dialogues with the users. It is equally important that the system not be limited to querying for individual facts—as is the case, for example, with database query systems. Rather this research provides a base for constructing systems that can engage in extended interactions to determine what a user intends (including when this differs from what he literally requests), and that provide responses appropriate to a particular user and discourse situation. Among the core capabilities of these systems are those for reasoning about the knowledge, goals, and plans of other agents where these other agents may be users or other computer systems.

This report covers work done on the KLAUS (Knowldge-Learning and Using System) project during the period July 1982 to January 1983. We were then engaged in developing two aspects of KLAUS: the natural-language-processing component and the deduction system component.

Section 2 (by Stuart M. Shieber) discusses a parsing technique whose behavior resembles human behavior in the intepretation of certain problematical types of sentences. Native speakers of English show definite and consistent preferences for certain readings of syntactically ambiguous sentences. A user of a natural-language-processing system would naturally expect it to reflect the same preferences. Thus, such systems must model in some way the *linguistic performance* as well as the *linguistic competence* of the native speaker. We have developed a variant of the LALR(1) shift-reduce algorithm that models the preference behavior of

native speakers for a range of syntactic-preference phenomena reported in the psycholinguistic literature, including the recent data on lexical preferences. The algorithm yields the preferred parse deterministically, without building multiple parse trees and choosing among them. As a side effect, it displays appropriate behavior in processing the much discussed garden-path sentences. The parsing algorithm has been implemented and has confirmed the feasibility of our approach to the modeling of these phenomena.

Section 3 (by Mark E. Stickel) discusses elements of the incorporation of equational and nonequational theories into the nonclausal connection-graph resolution theorem-proving program being used as the KLAUS deduction system. Incorporation of theories at a low level in the deduction system by means of simplification, special unification, and extended matching can reduce the size of the search space and the length of proofs, thereby substantially increasing the effectiveness of the deduction system. Proofs may also be more natural and readable. Three main techniques are presently being used for building in theories. Demodulation and special unification are used for building in equational theories. Demodulation is extended to simplify atomic formulas as well as terms allowing fast mandatory unit resolution and subsumption. Special unification is used to implement associative and/or commutative functions with or without identity. A coding trick is used to require a variable to match a single subterm of an associative term, thus permitting selection of single elements of sets without the need to make a distinction between simple and fragment variables. Nonequational theories such as partial or total orderings and taxonomic relationships can also be built in by extending the matching process. Unlike the extension of unification for equational theories, building in nonequational theories requires that the matching process take account of the polarities of the matched wffs and possibly return a wff as a result along with a substitution. This can be implemented by controlled hyperresolution, in which hyperresolution-like operations are specified for wffs of the theory while ordinary resolution operations are employed elsewhere.

2. Sentence Disambiguation by a Shift-Reduce Parsing Technique

2.1. Introduction

For natural-language-processing systems to be useful, they must assign the same interpretation to a given sentence that a native speaker would, since that is precisely the behavior users will expect. Consider, for example, the case of ambiguous sentences. Native speakers of English show definite and consistent preferences for certain readings of syntactically ambiguous sentences [Ki73, FF78, FBK82]. A user of a natural-language-processing system would naturally expect it to reflect the same preferences. Thus, such systems must model in some way the *linguistic performance* as well as the *linguistic competence* of the native speaker.

This idea is certainly not new in the artificial-intelligence literature. The pioneering work of Marcus [Ma80] is perhaps the best known example of linguistic-performance modeling in AI. Starting from the hypothesis that "deterministic" parsing of English is possible, he demonstrated that certain performance constraints, e.g., the difficulty of parsing garden-path sentences, could be modeled. His claim about deterministic parsing was quite strong. Not only was the behavior of the parser required to be deterministic, but, as Marcus claimed,

> The interpreter cannot use some general rule to take a nondeterministic grammar specification and impose arbitrary constraints to convert it to a deterministic specification (unless, of course, there is a general rule which will always lead to the correct decision in such a case). [Ma80, p.14]

We have developed and implemented a parsing system that, given a nondeterministic grammar, forces disambiguation in just the manner Marcus rejected (i.e. through general rules); it thereby exhibits the same preference behavior that psycholinguists have attributed to native speakers of English for a certain range of ambiguities. These include structural ambiguities [FF78, FF80, Wa80] and lexical preferences [FBK82], as well as the garden-path sentences as a side effect. The parsing system is based on the shift-reduce scheduling technique of Pereira [Pe82].

Our parsing algorithm is a slight variant of LALR(1) parsing and, as such, exhibits the three conditions postulated by Marcus for a deterministic mechanism: it is data-driven, reflects expectations, and has look-ahead. Like Marcus's parser, our parsing system is deterministic. Unlike Marcus's parser, the grammars used by ours can be ambiguous.

2.2. The Phenomena to be Modeled

The parsing system was designed to manifest preferences among structurally distinct parses of ambiguous sentences. It does this by building just one parse tree—rather than building multiple parse trees and choosing among them. Like the Marcus parsing system, ours does not do disambiguation requiring "extensive semantic processing," but, in contrast to Marcus, it does handle such phenomena as PP-attachment insofar as there exist *a priori*, preferences for one attachment over another. By *a priori* we mean preferences that are exhibited in contexts where pragmatic or plausibility considerations do not tend to favor one reading over the other. Rather than make such value judgments ourselves, we defer to the psycholinguistic literature (specifically [FF78], [FF80] and [FBK82]) for our examples.

The parsing system models the following phenomena:

Right Association Native speakers of English tend to prefer readings in which constituents are "attached low." For instance, in the sentence

```
Joe bought the book that I had been trying
to obtain for Susan.
```

the preferred reading is one in which the prepositional phrase "for Susan" is associated with "to obtain" rather than "bought."

Minimal Attachment On the other hand, higher attachment is preferred in certain cases such as

```
Joe bought the book for Susan.
```

in which "for Susan" modifies "the book" rather than "bought." Frazier and Fodor [FF78] note that these are cases in which the higher attachment includes fewer nodes in the parse tree. Our analysis is somewhat different.

Lexical Preference Ford *et al.* [FBK82] present evidence that attachment preferences depend on lexical choice. Thus, the preferred reading for

```
The woman wanted the dress on that rack.
```

has low attachment of the PP, whereas

The woman positioned the dress on that rack.

has high attachment.

Garden-Path Sentences

Grammatical sentences such as

The horse raced past the barn fell.

seem actually to receive no parse by the native speaker until some sort of "conscious parsing" is done. Following Marcus [Ma80], we take this to be a hard failure of the human sentence-processing mechanism.

It will be seen that all these phenomena are handled in our parser by the same general rules. The simple context-free grammar used[1] (see Appendix I) allows both parses of the ambiguous sentences as well as one for the garden-path sentences. The parser disambiguates the grammar and yields only the preferred structure. The actual output of the parsing system can be found in Appendix II.

2.3. The Parsing System

The parsing system we use is a shift-reduce parser. Shift-reduce parsers [AJ74] are a very general class of bottom-up parsers characterized by the following architecture. They incorporate a *stack* for holding constituents built up during the parse and a *shift-reduce table* for guiding the parse. At each step in the parse, the table is used for deciding between two basic types of operations: the *shift* operation, which adds the next word in the sentence (with its preterminal category) to the top of the stack, and the *reduce* operation, which removes several elements from the top of the stack and replaces them with a new element—for instance, removing an NP and a VP from the top of the stack and replacing them with an S. The *state* of the parser is also updated in accordance with the shift-reduce table at each stage. The combination of the stack, input, and state of the parser will be called *a configuration* and will be notated as, for example,

[1] We make no claims as to the accuracy of the sample grammar, which is obviously a gross simplification of English syntax. Its role is merely to show that the parsing system is able to disambiguate the sentences under consideration correctly.

stack: NP V	*input:* Mary	*state:* 10

where the stack contains the nonterminals NP and V, the input contains the lexical item Mary, and the parser is in state 10.

By way of example, we demonstrate the operation of the parser (using the grammar of Appendix I) on the oft-cited sentence "John loves Mary." Initially the stack is empty and no input has been consumed. The parser begins in state 0.

stack:	*input:* John loves Mary	*state:* 0

As elements are shifted to the stack, they are replaced by their preterminal category.[2] The shift-reduce table for the grammar of Appendix I states that in state 0, with a proper noun as the next word in the input, the appropriate action is a shift. The new configuration, therefore, is

stack: PNOUN	*input:* loves Mary	*state:* 4

The next operation specified is a reduction of the proper noun to a noun phrase, yielding

stack: NP	*input:* loves Mary	*state:* 2

The verb and second proper noun are now shifted, in accordance with the shift-reduce table, thus exhausting the input, and the proper noun is then reduced to an NP.

stack: NP V	*input:* Mary	*state:* 10
stack: NP V PNOUN	*input:*	*state:* 4
stack: NP V NP	*input:*	*state:* 14

Finally, the verb and noun phrase on the top of the stack are reduced to a VP

stack: NP VP	*input:*	*state:* 6

which is in turn reduced, together with the subject NP, to an S.

---

[2]But see Section 2.3.2 for an exception.

stack: S	*input:*	*state:* 1

This final configuration is an accepting configuration, since all the input has been consumed and an S derived. Thus, the sentence is grammatical according to the grammar of Appendix I, as expected.

2.3.1 Differences from the Standard LR Techniques

The shift-reduce table mentioned above is generated automatically from a context-free grammar by the standard algorithm [AJ74]. The parsing algorithm differs, however, from the standard LALR(1) parsing algorithm in two ways. First, instead of assigning preterminal symbols to words as they are shifted, the algorithm allows the assignment to be delayed if the word is ambiguous among preterminals. When the word is used in a reduction, the appropriate preterminal is assigned.

Second, and most importantly, since true LR parsers exist only for unambiguous grammars, the normal algorithm for deriving LALR(1) shift-reduce tables yields a table that may specify conflicting actions under certain configurations. It is through the choice made from the options in a conflict that the preference behavior we desire is engendered.

2.3.2 Preterminal Delaying

One key advantage of shift-reduce parsing that is critical in our system is the fact that decisions about the structure to be assigned to a phrase are postponed as long as possible. In keeping with this general principle, we extend the algorithm to allow the assignment of a preterminal category to a lexical item to be deferred until a decision is forced upon it, so to speak, by an encompassing reduction. For instance, we would not want to decide on the preterminal category of the word "that," which can serve as either a determiner (DET) or complementizer (THAT), until some further information is available. Consider the sentences

```
That problem is important.
```

```
That problems are difficult to solve is im-
portant.
```

Instead of assigning a preterminal to "that," we leave open the possibility of assigning either DET or THAT until the first reduction that involves the word. In the first case, this reduction will be by the rule NP → DET NOM, thus forcing, once and for all, the assignment of DET as preterminal. In the second case, the DET NOM analysis is disallowed on the basis of number agreement, so that the first applicable reduction is the COMP S reduction to $\overline{S}$, forcing the assignment of THAT as preterminal.

Of course, the question arises as to what state the parser goes into after shifting the lexical item "that." The answer is quite straightforward, though its interpretation *vis à vis* the determinism hypothesis is subtle. The simple answer is that the parser enters into a state corresponding to the union of the states entered upon shifting a DET and upon shifting a THAT, respectively, in much the same way as the deterministic simulation of a nondeterministic finite automaton enters a "union" state when faced with a nondeterministic choice. Are we then merely simulating a nondeterministic machine here? The answer is equivocal. Although the implementation acts as a simulator for a nondeterministic machine, the nondeterminism is *a priori* bounded, given a particular grammar and lexicon.[3] Thus, the nondeterminism could be traded in for a larger, albeit still finite, set of states, unlike the nondeterminism found in other parsing algorithms. Another way of looking at the situation is to note that there is no observable property of the algorithm that would distinguish the operation of the parser from a deterministic one. In some s there is no interesting difference between the limited nondeterminism of this parser and Marcus's notion of strict determinism. In fact, the implementation of Marcus's parser also embodies a bounded nondeterminism in much the same way this parser does.

The property that discriminates between this parser and that of Marcus is a slightly different one, namely, the property of *quasi-real-time operation*.[4] By quasi-real-time operation,

---

[3]The boundedness comes about because only a finite amount of information is kept per state (an integer) and the nondeterminism stops at the preterminal level, so that the splitting of states does not propagate.

[4]I am indebted to Mitch Marcus for this observation and the previous comparison with his parser.

Marcus means that there exists a maximum interval of parser operation for which no output can be generated. If the parser operates for longer than this, it must generate some output. For instance, the parser might be guaranteed to produce output (i.e., structure) at least every three words. However, because preterminal assignment can be delayed indefinitely in pathological grammars, there may exist sentences in such grammars for which arbitrary numbers of words need to be read before output can be produced. It is not clear whether this is a real disadvantage or not, and, if so, whether there are simple adjustments of the algorithm that would result in quasi-real-time behavior. In fact, it is a property of bottom-up parsing in general that quasi-real-time behavior is not guaranteed. Our parser has a less restrictive but similar property, *fairness*—namely, our parser generates output that is linear in the input, though there is no constant over which output is guaranteed. For a fuller discussion of these properties, see Pereira and Shieber [PS-].

To summarize, preterminal delaying, as an intrinsic part of the algorithm, does not actually change the basic properties of the algorithm in any observable way. Note, however, that preterminal assignments, like reductions, are irrevocable once they have been made (as a by-product of the algorithm's determinism). Such decisions can therefore lead to garden paths, as they do for the sentences presented in Section 2.3.6.

We now discuss the central feature of the algorithm, namely, the resolution of shift-reduce conflicts.

2.3.3 The Disambiguation Rules

Conflicts arise in two ways: *shift-reduce* conflicts, in which the parser has the option of either shifting a word onto the stack or reducing a set of elements on the stack to a new element; *reduce-reduce* conflicts, in which reductions by several grammar rules are possible. The parser uses two rules to resolve these conflicts:[5]

[5]The original notion of using a shift-reduce parser and general scheduling principles to handle right association and minimal attachment, together with the following two rules, are due to Fernando Pereira [Pe82]. The formalization of preterminal delaying and the extensions to the lexical-preference cases and garden-path behavior are due to the author.

(1) Resolve shift-reduce conflicts by shifting.

(2) Resolve reduce-reduce conflicts by performing the longer reduction.

These two rules suffice to engender the appropriate behavior in the parser for cases of right association and minimal attachment. Though we demonstrate our system primarily with PP-attachment examples, we claim that the rules are generally valid for the phenomena being modeled [PS–].

2.3.4 Some Examples

Some examples demonstrate these principles. Consider the sentence

Joe took the book that I bought for Susan.

After a certain amount of parsing has been completed deterministically, the parser will be in the following configuration:

stack: NP V NP that NP V	*input:* for Susan	*state:* 23

with a shift-reduce conflict, since the V can be reduced to a VP/NP[6] or the P can be shifted. The principles presented would solve the conflict in favor of the shift, thereby leading to the following derivation:

stack: NP V NP that NP V P	*input:* Susan	*state:* 12
stack: NP V NP that NP V P NP	*input:*	*state:* 19
stack: NP V NP that NP V PP	*input:*	*state:* 24
stack: NP V NP that NP VP/NP	*input:*	*state:* 22
stack: NP V NP that S/NP	*input:*	*state:* 16
stack: NP V NP $\overline{S}$	*input:*	*state:* 7
stack: NP V NP	*input:*	*state:* 14

[6]The "slash-category" analysis of long-distance dependencies used here is loosely based on the work of Gazdar [Ga81]. The Appendix I grammar does not incorporate the full range of slashed rules, however, but merely a representative selection for illustrative purposes.

stack: NP VP	*input:*	*state:* 6
stack: S	*input:*	*state:* 1

which yields the structure

$$[_S \text{Joe}[_{VP} \text{took}[_{NP}[_{NP} \text{the book}][_{\overline{S}} \text{that I bought for Susan}]]]]$$

The sentence

Joe bought the book for Susan.

demonstrates resolution of a reduce-reduce conflict. At some point in the parse, the parser is in the following configuration:

stack: NP V NP PP	*input:*	*state:* 20

with a reduce-reduce conflict. Either a more complex NP or a VP can be built. The conflict is resolved in favor of the longer reduction, i.e., the VP reduction. The derivation continues:

stack: NP VP	*input:*	*state:* 6
stack: S	*input:*	*state:* 1

ending in an accepting state with the following generated structure:

$$[_S \text{Joe}[_{VP} \text{bought}[_{NP} \text{the book}][_{PP} \text{for Susan}]]]$$

2.3.5 Lexical Preference

To handle the lexical-preference examples, we extend the second rule slightly. Preterminal-word pairs can be stipulated as either *weak* or *strong*. The second rule becomes

(2) Resolve reduce-reduce conflicts by performing the longest reduction *with the strongest leftmost stack element.*[7]

Therefore, if it is assumed that the lexicon encodes the information that the triadic form of "want" (V2 in the sample grammar) and the dyadic form of "position" (V1) are both weak, we can see the operation of the shift-reduce parser on the "dress on that rack" sentences of Section 2. Both sentences are similar in form and will thus have a similar configuration when the reduce-reduce conflict arises. For example, the first sentence will be in the following configuration:

stack: NP wanted NP PP	*input:*	*state:* 20

In this case, the longer reduction would require assignment of the preterminal category V2 to "want," which is the weak form; thus, the shorter reduction will be preferred, leading to the derivation

stack: NP wanted NP	*input:*	*state:* 14
stack: NP VP	*input:*	*state:* 6
stack: S	*input:*	*state:* 1

and the underlying structure

$$[_S \text{the woman} [_{VP} \text{wanted} [_{NP} [_{NP} \text{the dress}] [_{PP} \text{on that rack}]]]]$$

In the case in which the verb is "positioned," however, the longer reduction does not yield the weak form of the verb; it will therefore be invoked, resulting in the structure

$$[_S \text{the woman} [_{VP} \text{positioned} [_{NP} \text{the dress}] [_{PP} \text{on that rack}]]]$$

[7]Note that strength takes precedence over length.

2.3.6 Garden-Path Sentences

As a side effect of these conflict resolution rules, certain sentences in the language of the grammar will receive no parse by the parsing system just discussed. These sentences are apparently the ones classified as "garden-path" sentences, a class that humans also have great difficulty parsing. Marcus's conjecture that such difficulty stems from a hard failure of the normal sentence-processing mechanism is directly modeled by the parsing system presented here.

For instance, the sentence

```
The horse raced past the barn fell.
```

exhibits a reduce-reduce conflict before the last word. If the participial form of "raced" is weak, the finite verb form will be chosen; consequently, "raced past the barn" will be reduced to a VP rather than a participial phrase. The parser will fail shortly, since the correct choice of reduction was not made.

Similarly, the sentence

```
That scaly, deep-sea fish should be under-
water is important.
```

will fail, though grammatical. Before the word "should" is shifted, a reduce-reduce conflict arises in forming an NP from either "That scaly, deep-sea fish" or "scaly, deep-sea fish." The longer (incorrect) reduction will be performed and the parser will fail.

Other examples, e.g., "the boy got fat melted," or "the prime number few" would be handled similarly by the parser, though the sample grammar of Appendix I does not parse them [PS-].

2.4. Conclusion

To be useful, natural-language systems must model the behavior, if not the method, of the native speaker. We have demonstrated that a parser using simple general rules

for disambiguating sentences can yield appropriate behavior for a large class of performance phenomena—right association, minimal attachment, lexical preference, and garden-path sentences—and that, morever, it can do so deterministically without generating all the parses and choosing among them. The parsing system has been implemented and has confirmed the feasibility of our approach to the modeling of these phenomena.

Appendix I. The Test Grammar

The following is the grammar used to test the parsing system descibed in the paper. Not a robust grammar of English by any means, it is presented only for the purpose of establishing that the preference rules yield the correct results.

S → NP VP	VP → AUX VP	$\bar{S}$/NP → that S/NP
S → $\bar{S}$ VP	VP → V0	S/NP → VP
NP → DET NOM	VP → V1 NP	S/NP → NP VP/NP
NP → NOM	VP → V2 NP PP	VP/NP → V1
NP → PNOUN	VP → V3 INF	VP/NP → V2 PP
NP → NP $\bar{S}$/NP	VP → V4 ADJ	VP/NP → V3 INF/NP
NP → NP PARTP	VP → V5 PP	VP/NP → AUX VP/NP
NP → NP PP	$\bar{S}$→ that S	INF/NP → to VP/NP
DET → NP 's	INF → to VP	
NOM → N	PP → P NP	
NOM → ADJ NOM	PARTP → VPART PP	

Appendix II. Sample Runs

```
>> Joe bought the book that I had been trying to obtain for Susan

Accepted: [s
           (np (pnoun Joe))
           (vp
             (v1 bought)
             (np
               (np (det the)
```

```
                (nom (n book)))
            (sbar/np
              (that that)
              (s/np
                (np (pnoun I))
                (vp/np
                  (aux had)
                  (vp/np (aux been)
                         (vp/np (v3 trying)
                                (inf/np (to to)
                                        (vp/np (v2 obtain)
                                               (pp (p for)
                                                   (np (pnoun Susan]
```

```
>> Joe bought the book for Susan

Accepted: [s (np (pnoun Joe))
             (vp (v2 bought)
                 (np (det the)
                     (nom (n book)))
                 (pp (p for)
                     (np (pnoun Susan]
```

```
>> The woman wanted the dress on that rack

Accepted: [s (np (det The)
                 (nom (n woman)))
             (vp (v1 wanted)
                 (np (np (det the)
                         (nom (n dress)))
                     (pp (p on)
                         (np (det that)
                             (nom (n rack]
```

```
>> The woman positioned the dress on that rack

Accepted: [s (np (det The)
                 (nom (n woman)))
             (vp (v2 positioned)
                 (np (det the)
                     (nom (n dress)))
                 (pp (p on)
                     (np (det that)
                         (nom (n rack]
```

```
>> The horse raced past the barn fell
```

```
Parse failed.  Current configuration:
state:  (1)

stack:  <(0)>       [s (np (det The)
                       (nom (n horse)))
                    (vp (v5 raced)
                       (pp (p past)
                          (np (det the)
                             (nom (n barn]


input:  (v0 fell)
        (end)

>> That scaly deep-sea fish should be underwater is important

Parse failed.  Current configuration:
state:  (1)

stack:  <(0)>       [s [np (det That)
                       (nom (adj scaly)
                          (nom (adj deep-sea)
                             (nom (n fish]
                    (vp (aux should)
                       (vp (v4 be)
                          (adj underwater]


input:  (v4 is)
        (adj important)
        (end)
```

3. Building in Equational and Nonequational Theories

3.1. Introduction

This section describes continuing work on a nonclausal connection-graph resolution theorem-proving program for the first-order predicate calculus [St82]. The dominant concern in this research is to develop computationally effective reasoning techniques for concepts that are embedied in natural-language-understanding applications.

Using a general purpose theorem-proving program with an axiomatization of the domain for a natural-language-understanding application is unlikely to yield acceptable performance in the absence of specifications detailing how the axioms are to be used. Our current theorem-proving program has facilities for heuristic search and some control specification expressed by the use of special logical connectives [Mo82]. We have now added to the program a number of other facilities to improve performance that complement the heuristic search and control specification facilities. Demodulation is used to simplify terms and special unification is used to build in associativity and commutativity. We are also developing extended matching techniques for building in nonequational theories, just as demodulation and special unification are used for building in equational theories [Pl72].

3.2. Demodulation

Demodulation [WR67] is the powerful technique of keeping all terms simplified with respect to a list of ordered equalities called demodulators. This reduces the size of the terms and, because equivalent terms may be simplified to the same term, facilitates subsumption. (A complete set of reductions [KB70,La75,Hue80,Hul80,PS81] is essentially a list of demodulators that is *guaranteed* to simplify equivalent terms to the same term.) Demodulation can also be used as a programming mechanism for a variety of purposes [WW82], such as counting symbol occurrences in expressions, classifying expressions, etc.

We extend demodulation slightly beyond its conventional usage by permitting atomic formulas as well as terms to be simplified. In principle, demodulators can be written to perform arbitrary simplifications. We allow term→term, atom→atom, and atom→truth-value simplifications. Simplifications of the form atom→term, etc. are illogical; atom→wff simplifications are useful (e.g., for expanding definitions) but presently unimplemented; it is expensive to check the applicability of wff→wff simplifications and many useful cases can be anticipated and programmed directly (e.g., $A \wedge \neg A \rightarrow false$.)

Simplifications of the form atom→truth-value are particularly useful. For example, if $A \rightarrow true$ and $B \rightarrow false$ are demodulators, the clauses $\neg A \vee C$ and $B \vee C$ can be simplified to C and $A \vee C$ and $\neg B \vee C$ can be simplified to *true*—the effect is either that of a mandatory unit resolution operation plus subsumption of the parent clause or (using tautology elimination) of subsumption of the clause by a unit.

Such immediate, mandatory resolution and subsumption operations without retention of intermediate results account for much of the success of predicate calculus theorem proving by the Knuth-Bendix procedure [KB70,La75,Hue81,Hul81,PS81] with a complete set of reductions for Boolean algebra [Hs81].

Demodulation provides a mechanism for implementing procedural attachment: the right-hand side of the demodulator specifies the compututation of a new expression for all expressions matching the left-hand side. This can be used to incorporate numerical computations. (Winker and Wos [WW82] also use demodulation to perform arithmetic operations, notably in counting demodulators.) Furthermore, it can be used to attach code that imposes constraints, such as requiring a set of variables to have distinct values.

Forward demodulation is the use of demodulators to simplify wffs derived after the demodulator was created. Backward demodulation is the use of a newly added demodulator to simplify previously existing wffs. Only forward demodulation is used at present. Backward demodulation appears to be significantly less effective than forward demodulation for the following reasons:

- The most effective demodulators are often provided before the start of the proof and

will thus be available for use in forward demodulation. Only demodulators generated in the course of the proof can be used for backward demodulation; these are often more specific and less powerful than those initially provided.

- If a wff that can be, but is not, backward-demodulated is resolved upon, then, except for the atom resolved upon, instances of all the atoms that can be backward-demodulated will appear in the resolvent and can be forward-demodulated as well. Thus, although the wff is not demodulated by a new demodulator, its descendants derived after the addition of the new demodulator will be.

3.3. Special Unification

The incorporation of concepts such as associativity, commutativity, and idempotence into the unification algorithm has great potential for eliminating explicit equality reasoning, facilitating subsumption by recognizing nonidentical but equivalent terms, and generally reducing the size of the search space. Among the many special unification algorithms developed [RS79], the most pervasively useful ones are those for associativity and/or commutativity with or without identity [LS75, LS76, Si75, Si76, St77, St81], and it is these that have so far been implemented for this theorem prover.

Using special unification to handle such concepts as associativity and commutativity is clearly the best available approach. Alternatives require axiomatizing the properties using the equality relation, e.g., $f(x,y) = f(y,x)$ and $f(f(x,y),z) = f(x,f(y,z))$, or using some alternative encoding of the function, e.g., $P(x,y,z) \supset P(y,x,z)$ and $P(x,y,u) \wedge P(y,z,v) \wedge P(u,z,w) \supset P(x,v,w)$ (and $P(x,y,u) \wedge P(y,z,v) \wedge P(x,v,w) \supset P(u,z,w)$ for the "other half" of the associative law), where $P(x,y,z)$ means $f(x,y) = z$.

Using the equality relation to axiomatize such properties requires that equality reasoning be employed in the deduction system. In many instances, associativity and commutativity might provide the only requirement for equality reasoning. Because reasoning about equality by using resolution plus the equality axioms is notoriously inefficient, and even using special

equality inference rules often results in an unmanageably large search space, it would be prudent to avoid equality-based axiomatizations of such concepts.

Using alternative encodings, such as $P(x, y, z)$ for $f(x, y) = z$, often does result in greater efficiency than using equality based axiomatizations, but the alternative encoding obscures the meaning of the wffs, and the search space is still likely to be quite large because of the universal applicability of assertions containing atoms like $P(x, y, z)$, which match any complementary literal with predicate symbol P.

Because associativity and/or commutativity are not built in primitively, the system does not recognize that, for example, $f(a, f(b, c))$, $f(a, f(c, b))$, $f(f(b, c), a)$, $f(f(c, b), a)$, $f(b, f(a, c))$, ..., (the equivalent set of expressions encoded using the P predicate looks even worse) are all equivalent and it is sufficient to use only one of them.

Building in properties like associativity and commutativity solves these difficulties. Although the branching factor of special unification algorithms is greater than that for ordinary unification (which always returns either no unifier or a single unifier whereas special unification may return an arbitrary number of unifiers), the situation is still a vast improvement over the extremely redundant, high-branching-factor search space present when associativity and commutativity are axiomatized.

Building in associativity and commutativity has usefulness well beyond the obvious mathematical applications such as handling the symmetry of equality and the associativity and commutativity of addition and multiplication. A nonmathematical example of the usefulness of building associativity and commutativity into the unification algorithm, suggested by use of the theorem prover in natural-language-understanding applications, is the use of $\neg T(K(x, w, n), p) \supset \neg T(K(Ker(x, y), w, n), p)$ written as part of a formulation of mutual knowledge [Ap82]. It roughly states that, if proposition p is not known by agent x, p is not common knowledge of agents x and y. If this implication is applied in a forward direction, it generates an infinite sequence of results *unless* Ker is treated as associative and commutative—in which case the first application of the axiom results in a formula that subsumes all the rest, eliminating an infinite branch in the search space.

Sets can be represented by using the associative-commutative function *set* with identity: $\{a, b, c\}$ is represented by $set(a, b, c)$. The idempotence of sets can be handled either by (1) using a demodulator $set(x, x, y) = set(x, y)$, which eliminates redundant elements, or (2) declaring *set* to be idempotent. The former is easier and is preferable if, in unifying $\{x, y\}$ and $\{a, b, c\}$, x and y are to be assigned disjoint sets of elements. However, if x and y must be assigned all sets of values such that their union is $\{a, b, c\}$, then *set* should be declared to be idempotent, and associative-commutative-idempotent unification with identity [LS76] should be used.

It is often necessary to select a single element of a set rather than (as in unifying $\{x, y\}$ and $\{a, b, c\}$) decompose a set into two parts, neither of which is required to be a single element. In AI programming languages [He72,RD72], this problem is solved by making a distinction among variables: simple variables match single elements; fragment variables can match zero or more. If the set $\{a, b, c\}$ is encoded as $set(el(a), el(b), el(c))$, this distinction is unnecessary. If a variable is required to match a single element of the set, it is enclosed in *el*. For example, $x \in \{a, b, c\}$ can be expressed by $in(x, set(el(a), el(b), el(c)))$ and single element values for x can be nondeterministically selected by unification with the axiom $in(x, set(el(x), y))$.

In our implementation of associative and/or commutative unification with or without identity, we have imposed some restrictions on the unification algorithm that result in some loss of logical completeness.

First, we constrain associative unification to not return all possible unifier because there may be an infinite number of them, as in the case of unifying $f(a, x)$ and $f(x, a)$ with unifiers $x \leftarrow a$, $x \leftarrow f(a, a)$, ... Variables are constrained to match sequences of arguments of the other term; variables are not split so that, e.g., $f(a, x)$ and $f(y, b)$ can be matched with $x \leftarrow f(x_1, x_2), x_2 \leftarrow b, y \leftarrow f(a, x_1)$ (the value of the variable x is split between y and b). Completeness can be recovered by augmenting this incomplete unification with a variable-splitting operation that replaces a variable x by $f(x_1, x_2)$.

The second restriction we impose is that we invoke these special unification algorithms only if the head (predicate or function) symbols of the expressions match. Thus, although $f(x, y)$

and 0 match where 0 is the identity for f (with $x \leftarrow 0, y \leftarrow 0$), this fact will be undetected because the function symbols of the two terms do not match.

3.4. Extended Matching for Nonequational Theories

Properties like associativity and commutativity are expressed by equational theories, i.e., lists of equalities or equivalences. It would be useful to apply techniques that are similar to special unification to nonequational theories, e.g., lists of implications.

The purpose of extended matching for nonequational theories is to extend the matching process used in resolution (the process that ordinarily unifies the arguments of complementary literals, i.e., literals with opposite polarity, one negated, the other unnegated, with the same predicate symbol) to a process that matches literals that are complementary according to the axioms being incorporated into the matching process. Use of the extended matching process would eliminate the requirement for explicitly using the built-in axioms. Here are some examples of nonequational theories that occur in natural-language-understanding and many other applications and that we would wish to build in for efficiency:

- Subtype relations. The assertion $\forall x.elephant(x) \supset mammal(x)$ can be incorporated into the matching process if matching is permitted between positive occurrences of the *elephant* predicate and negative occurrences of the *mammal* predicate. Thus, $elephant(Clyde)$ and $\neg mammal(Clyde)$ could be treated as complementary literals. Building in the assertion $\forall x.mammal(x) \supset animal(x)$ as well would also permit positive occurrences of *elephant* or *mammal* to be matched with negative occurrences of *animal*.

- Properties. The assertion $\forall x.elephant(x) \supset color\text{-}of(x, gray)$ can be eliminated if matching is permitted between positive occurrences of the *elephant* predicate and negative occurrences of the *color-of* predicate, with its second argument required to match *gray*.

- Disjointness. The assertion $\forall x.\neg man(x) \vee \neg woman(x)$ can be eliminated if matching is permitted between positive occurrences of the *man* predicate and positive occurrences of the *woman* predicate. This is an example of a situation in which literals with the

same polarity, e.g., $man(John)$ and $woman(John)$, both unnegated, are complementary literals in the theory.

Similarly, the assertion $\forall x \forall y. \neg x < y \vee \neg x > y$ can be eliminated if matching is permitted between positive occurrences of the $<$ predicate and positive occurrences of the $>$ predicate.

- Stronger/weaker predicates. The assertion $\forall x \forall y. on(x,y) \supset above(x,y)$ can be eliminated if matching is permitted between positive occurrences of the *on* predicate and negative occurrences of the *above* predicate.

  Similarly, the assertion $\forall x \forall y. x < y \supset x \leq y$ can be eliminated if matching is permitted between positive occurrences of the $<$ predicate and negative occurrences of the $\leq$ predicate.

- Permuted arguments. The assertion $\forall x \forall y. x = y \supset y = x$ can be eliminated if matching is permitted between positive occurrences of the $=$ predicate and negative occurrences of the $=$ predicate with its arguments permuted.

  The assertion $\forall x \forall y \forall z. M(x,y,z) \supset M(y,x,z)$, where $M(x,y,z)$ means $x \times y = z$, can be eliminated if matching is permitted between positive occurrences of the M predicate and negative occurrences of the M predicate with its first two arguments permuted.

  The assertion $\forall x \forall y. x < y \equiv y > x$ can be eliminated if matching is permitted between occurences of the $<$ and $>$ predicates with reversed arguments and opposite polarities.

  The $=$ and M predicate examples can also be handled by commutative unification.

- Use of residue. The assertion $\forall x \forall y \forall z. x < y \wedge y < z \supset x < z$ can be partly built in by allowing inferences such as "from $a < b$ and $b < c$ infer $a < c$". In this case, $a < b$ and $b < c$ are complementary literals, provided $\neg a < c$ is true. The negation of this condition is called the residue and is disjoined to the resolvent. So, in this case, forming a resolvent entails matching and removing a pair of literals and disjoining the remaining literals plus the residue of the matching operation.

There are numerous advantages to incorporating these assertions into the matching

process:

- Deductions are shorter and easier to read. Even if the search space were the same size (exchanging breadth for depth), the reduced depth would still result in less work because (though the number of unifications may be comparable) the number of resolvents that have to be formed would decrease. The saving would be significant unless unification accounted for nearly all the effort in forming a resolvent.

- Incorporation of assertions like $\forall x \forall y . x = y \supset y = x$ in the matching process eliminates, for example, derivations such as $a = b, b = a, a = b, \ldots$, where (for efficiency) the second $a = b$ must either be eliminated (e.g., by subsumption) or not generated (as a result of restrictions imposed by the control strategy).

- The incorporation of a multistep implication chain in the matching process has the effect of performing the corresponding set of resolution operations on one of the wffs. In doing so, it effectively imposes a requirement that the atom resolved upon in each of these resolution operations be the atom introduced in the preceding resolution operation—an implicit ordering requirement that would otherwise have to be specified by the control strategy.

- If assertions are not incorporated into the matching process, performing a sequence of resolution operations on a wff with them may result in the generation of a pure atom (an atom that cannot be resolved with any other atom in the search space). At least the last step (and maybe more) of that sequence of operations was wasted effort. If the assertions are incorporated into the matching process, the match is successful if and only if the final atom produced in the sequence of resolution operations is not pure.

- In addition to the special case of not producing pure atoms (which have a branching factor of 0), incorporation of assertions into the matching process can (when the possible matches for an atom are counted) result in a cheaper or better estimate of the effective branching factor of the atom. If assertions are not incorporated in the matching process, branching-factor estimates are either cheap and unreliable (if they only count the number of immediate resolvents) or are expensive (if do tree searching to consider sequences

of resolution operations). Incorporation of the assertions into the matching process provides more reliable estimates at essentially the cost of the cheap estimate when the assertions are not incorporated.

- Because building axioms into extended matching requires the existence of two literals complementary in the theory, it can, for example, block the infinite derivation of $b < x \supset a < x$, $b < x \wedge x < y \supset a < y$, ..., from $a < b$, which is ordinarily derivable even if no other wffs mention a or b, which implies that no refutation is possible.

We are developing the theory of extending matching for the building in of axioms like those above. Extended matching is similar to Dixon's Z-resolution [Di73] and also to Harrison and Rubin's U-generalized resolution [HR78]; it should be an extension of both, however, because it does not restrict the built-in wffs to be length 2 clauses (as Z-resolution does) or be usable only when unit or input refutations exist, as is the case for U-generalized resolution.

We have also developed a prototype implementation of extended matching that is based on controlled hyperresolution. Controlled hyperresolution is essentially user-specified hyperresolution designating built-in wffs as the nucleus of hyperresolution operations and specifying what atoms of the wff are to be resolved on. Thus, building in $\forall x.elephant(x) \supset mammal(x)$ entails designating that wff as the nucleus for hyperresolution operations so that, for example, *false* is the hyperresolvent of $elephant(Clyde)$, $\neg mammal(Clyde)$, and $\forall x\ elephant(x) \supset mammal(x)$. Likewise, $a < c$ is a hyperresolvent of $a < b$, $b < c$ and the designated nucleus $\forall x \forall y \forall z.x < y \wedge y < z \supset x < z$.

References

[AJ74] Aho, A.V. and S.C. Johnson. LR parsing. *Computing Surveys 6*, 2 (1974), 99–124.

[Ap82] Appelt, D.E. Planning natural-language utterances to satisfy multiple goals. Technical Note 259, Artificial Intelligence Center, SRI International, Menlo Park, California, March 1982.

[Di73] Dixon, J.K. Z-resolution: theorem-proving with compiled axioms. *J. ACM 20,* 1 (January 1973), 127–147.

[FBK82] Ford, M., J. Bresnan, and R. Kaplan. A competence-based theory of syntactic closure. In J. Bresnan (ed.), *The Mental Representation of Grammatical Relations,* MIT Press, Cambridge, Massachusetts, 1982.

[FF78] Frazier, L. and J.D. Fodor. The sausage machine: a new two-stage parsing model. *Cognition 6* (1978), 291–325.

[FF80] Frazier, L. and J.D. Fodor. Is the human sentence parsing mechanism an ATN? *Cognition 8* (1980), 411–459.

[Ga81] Gazdar, G. Unbounded dependencies and coordinate structure, *Linguistic Inquiry 12,* (1981), 165–179.

[HR78] Harrison, M.C. and N. Rubin. Another generalization of resolution. *J. ACM 25,* 3 (July 1978), 341–351.

[He72] Hewitt, C. Description and theoretical analysis using schemata of PLANNER: a language for proving theorems and manipulating models in a robot. Artificial Intelligence Laboratory report AI TR-258, Massachusetts Institute of Technology, Cambridge, Massachusetts, April 1972.

[Hs81] Hsiang, J. Refutational theorem proving using term rewriting systems. Unpublished manuscript, Department of Computer Science, University of Illinois, Urbana, Illinois, July 1981.

[Hue80] Huet, G. Confluent reductions: Abstract properties and applications to term rewriting systems. *J. ACM 27,* 4 (October 1980), 797–821.

[Hul80] Hullot, J.M. A catalogue of canonical term rewriting systems. Technical Report CSL-113, Computer Science Laboratory, SRI International, Menlo Park, California, April 1980.

[Ki73] Kimball, J. Seven principles of surface structure parsing in natural language. *Cognition 2,* 1 (1973), 15–47.

[KB70] Knuth, D.E. and P.B. Bendix. Simple word problems in universal algebras. In J. Leech (ed.), *Computational Problems in Abstract Algebras,* Pergamon Press, 1970, pp. 263–297.

[La75] Lankford, D.S. Canonical inference. Report ATP-32, Department of Mathematics, University of Texas, Austin, Texas, December 1975.

[LS75] Livesey, M. and J. Siekmann. Termination and decidability results for string-unification. Memo CSM-12, Essex University Computing Center, Colchester, Essex, England, August 1975.

[LS76] Livesey, M. and J. Siekmann. Unification of A+C-terms (bags) and A+C+I-terms (sets). Interner Bericht Nr. 5/76, Institut für Informatik I, Universität Karlsruhe, Karlsruhe, West Germany, 1976.

[Ma80] Marcus, M. *A Theory of Syntactic Recognition for Natural Language*, MIT Press, Cambridge, Massachusetts, 1980.

[Mo80] Moore, R.C. Reasoning about knowledge and action. Technical Note 191, SRI Artificial Intelligence Center, October 1980.

[Pe82] Pereira, F.C.N. A new characterisation of attachment preferences. To appear in D. Dowty, L. Karttunen, and A. Zwicky (eds.) *Natural Language Processing. Psycholinguistic, Computational, and Theoretical Perspectives*, Cambridge University Press, Cambridge, England.

[PS-] Pereira, F.C.N. and S.M. Shieber. Shift-reduce scheduling and syntactic closure. To appear.

[PS81] Peterson, G.E. and M.E. Stickel. Complete sets of reductions for some equational theories. *J. ACM 28*, 2 (April 1981), 233–264.

[Pl72] Plotkin, G.D. Building-in equational theories. In Meltzer, B. and Michie, D. (eds.), *Machine Intelligence 7*, Halsted Press, 1972, pp. 73–90.

[RS79] Raulefs, P., J. Siekmann, P. Szabo, and E. Unvericht. A short survey on the state of the art in matching and unification problems. *SIGSAM Bulletin 13*, 2 (May 1979), 14–20.

[RD72] Rulifson, J.F., J.A. Derksen, and R.J. Waldinger. QA4: a procedural calculus for intuitive reasoning. Technical Note 73, Artificial Intelligence Center, SRI International, Menlo Park, California, November 1972.

[Si75] Siekmann, J. String-unification, part I. Essex University, Cochester, Essex, England, March 1975.

[Si76] Siekmann, J. T-unification, part I. Unification of commutative terms. Interner Bericht Nr. 4/76, Institut für Informatik I, Universität Karlsruhe, Karlsruhe, West Germany, 1976.

[St77] Stickel, M.E. Mechanical theorem proving and artificial intelligence languages. Ph.D. Dissertation, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, December 1977.

[St81] Stickel, M.E. A unification algorithm for associative-commutative functions. *J. ACM 28*, 3 (July 1981), 423–434.

[St82] Stickel, M.E. A nonclausal connection-graph resolution theorem-proving program. *Proceedings of the AAAI-82 National Conference on Artificial Intelligence*, Pittsburgh, Pennsylvania, August 1982, 229–233.

[Wa80] Wanner, E. The ATN and the sausage machine: which one is baloney? *Cognition 8,* (1980), 209–225.

[WW82] Winker, S.K. and L. Wos. Procedure implementation through demodulation and related tricks. *Proceedings of the 6th Conference on Automated Deduction,* New York, New York, June 1982, 109–131.

[WR67] Wos, L., G. Robinson, D. Carson, and L. Shalla. The concept of demodulation in theorem proving. *J. ACM 14,* 4 (October 1967), 698–709.

END

FILMED

9-83

DTIC